Enabling a lightweight Edge Gateway-as-a-Service for the Internet of Things

Roberto Morabito, Riccardo Petrolo, Valeria Loscrì, Nathalie Mitton

Enabling a lightweight Edge Gateway-as-a-Service for the Internet of Things

Roberto Morabito*, Riccardo Petrolo†, Valeria Loscrí†, and Nathalie Mitton†
*Ericsson Research, Jorvas, Finland - †Inria Lille - Nord Europe, France

***Abstract*—Since its introduction, the Internet of Things (IoT) has changed several aspects of our lives, leading to the commercialization of different heterogeneous devices. In order to bridge the gap among these heterogeneous devices, in some of the most common IoT use cases -e.g., smart home, smart buildings, etc.- the presence of a gateway as an enabler of interoperability is required. In this paper, we introduce the concept of a Gateway-as-a-Service (GaaS), a lightweight device that can be shared between different users thanks to the use of virtualization techniques. Performance has been evaluated on real hardware and results demonstrate the lightweight characteristics of the proposal.**

Keywords*—*Internet of Things, Edge Computing, Virtualization, Gateway, Container.

I. Introduction

The Internet of Things (IoT) is signing an important revolution in all the aspects of our lives, i.e., health, transportation, work, and so on and so forth. The tremendous interest behind the IoT has led to the development and commercialization of different types of devices -billions are already deployed, and the number will reach 50 billion by 2020 according to [1]-. These *objects* are often designed in order to better suit specific application needs; therefore by using various technologies in terms of hardware and network protocols. As a result, nowadays, we have a proliferation of heterogeneous *things* and *standards* that has brought to a fragmented and complex landscape. To bridge this gap, an interesting and promising approach is to apply semantic technologies -already well spread in the Web- as an enabler of interoperability among heterogeneous devices [2].

In some IoT contexts -such as smart home, smart buildings, smart farms, and so on and so forth- the interoperability among disparate devices can be achieved at the gateway. In [3] we proposed the architecture of a gateway capable to manage, on one hand, semantic-like things and, on the other hand, to act as an end-point for the presentation of data to users. In this paper, we introduce the concept of Gateway-as-a-Service (GaaS), as an efficient and lightweight device, which can be shared between different users. Thanks to emerging lightweight virtualization techniques, indeed, it is possible to provide high versatility and customizability to the platform. The proposal has been implemented on real hardware, and validated through an extensive performance evaluation.

The remainder of the paper is organized as follows. Section II explores related work. In Section III, we introduce the main characteristics of our Gateway-as-a-Service, while Section IV focuses on the performance evaluation and in particular the impact of using virtualization technologies in our solution. Section V concludes the paper.

This work is partially supported by CPER DATA, the FP7 VITAL project, the CAPES-COFECUB CROMO project, and by the FP7 Marie Curie METRICS project.

II. Related Work

In this section, we discuss related work linked to our proposal; we first analyze initiatives regarding the interoperability in the Internet of Things; then, we focus on the literature proposing solutions in which virtualization technologies are employed at the network edge.

As stated by Desai et al. in [4], a scalable IoT architecture should be independent from messaging protocol standards, while also providing integration and translation between various popular messaging protocols. Recently, semantic techniques started to became popular also in the IoT context [2], as an enabler of the interoperability among disparate devices. Authors in [4] describe a semantic IoT architecture where the gateway, located between physical level sensors and cloud-base services, provides translation between widely used CoAP, MQTT, and XMPP protocols; however authors do not focus on the interactions of the gateway with external users. Wu et al. in [5] propose the concept of Gateway-as-a-Service as a device that provides distributed cloud services for intelligent IoT applications based on the Web of Things (WoT). However, this solution does not consider the heterogeneity of different devices.

Bukhary et al. in [6] evaluate Docker container technology as a platform for Edge Computing. In this work, Docker has been evaluated in terms of deployment and termination, resource and service management, fault tolerance and caching. Authors' conclusions are that Docker represents a good solution to be employed in edge computing contexts. In our previous work [3], we include lightweight virtualization technologies in the design of a gateway for the Cloud of Things (IoT). In this work, the aforementioned idea is enhanced by introducing new features and an extensive performance evaluation. In [7], the design of an IoT gateway that can be efficiently employed also in an edge computing architecture has been proposed. In particular, that study shows how to efficiently and flexibly use Docker containers in order to customize the IoT platform, by offering data processing services. Container technologies are also used in a Capillary Network scenario [8], where Docker containers allow to package, deploy, and execute different functionality at the capillary gateway.

The aforementioned proposals strengthen our vision towards the implementation of an efficient and lightweight gateway for the Internet of Things. Thanks to the use of virtualization techniques, the gateway can be shared among different users and customized according to the use case needs.

III. Gateway-as-a-Service

The architecture of our Lightweight Edge Gateway-as-a-Service is depicted in Figure 1. The platform has been

designed to meet specific requirements: *(i)* interoperability; *(ii)* high energy-efficiency; *(iii)* fast allocation and flexibility in managing different services; *(iv)* isolation; *(v)* backup capabilities; *(vi)* multitenancy. In the following subsections, each key component is described in detail.

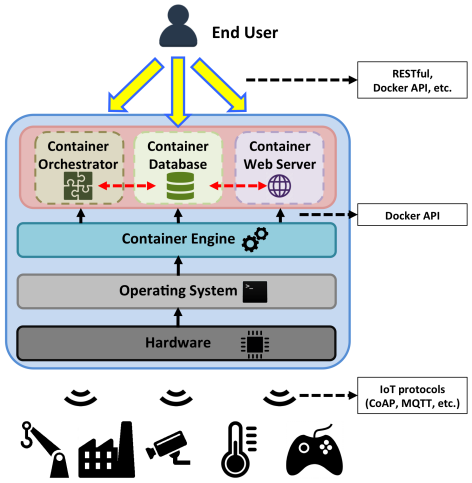


Fig. 1: Gateway Architecture.

A. Hardware and Operating System

ARM architecture is constantly becoming more widespread by virtue of its low-power characteristics and costs [9]. The Single Board Computer family Raspberry Pi (RPi) takes advantage of these features, and it became popular, over the last few years, thanks to the different purposes of use; the RPi finds room also in the IoT context, especially for providing gateway functionality [3].

In this paper, we selected the last two *generations* of the Raspberry Pi[1], the Raspberry Pi 2 (RPi2) model B[2] and the Raspberry Pi 3 (RPi3) model B[3], that has been released in February 2016. The main hardware characteristics of the two boards are summarized in Table I.

TABLE I: Raspberry Pi 2 and Raspberry Pi 3 features.

	Raspberry Pi 2 model B	**Raspberry Pi 3 model B**
Chipset	Broadcom BCM2836	Broadcom BCM2837
CPU	Quad-Core @900MHz	Quad-Core @1.2GHz
Memory	1GB LP-DDR2 400MHz	1GB LP-DDR2 900MHz
GPU	Broadcom VideoCore IV	Broadcom VideoCore IV
Ethernet	10/100 Mb/s	10/100 Mb/s
Flash Storage	MicroSD	MicroSD
Connectivity USB	4USB 2.0 Host	4USB 2.0 Host
OS	Linux, Windows 10	Linux, Windows 10
Price	$35	$35

As base **Operating System**, we use the image provided by Hypriot[4] running Raspbian Jessie with Linux kernel 4.4.10. Both RPis use 16 GB (*Transcend Premium* 400*x Class* 10 *UHS-I microSDHCTM*) memory card as storage device.

[1]https://www.raspberrypi.org/
[2]https://www.raspberrypi.org/products/raspberry-pi-2-model-b
[3]https://www.raspberrypi.org/products/raspberry-pi-3-model-b
[4]http://blog.hypriot.com

B. Container Virtualization Technology

According to some of the requirements listed at the beginning of this section, introducing lightweight virtualization technologies allows a system that benefits of interesting features such as: *(i)* fast building process, instantiation, and initialization of containers; *(ii)* high density of application/services due to the small container image; *(iii)* isolation between different instances. This is mainly due to the lightweight characteristics of container technologies if compared to alternative solutions such as hypervisor-based virtualization. The main difference between these two technologies are widely discussed in [10]. In our implementation, we use Docker[5] containers for executing the different instances. Docker introduces an underlying container engine, together with a functional API that allows easily building, management, and removal of a virtualized application. Docker version 1.11.0 has been used.

C. Application Components

As can be observed in Figure 1, from the application point of view, our architecture is characterized by three main components: *(i)* a web server that exposes services to the Internet -we chose **WildFly**[6], an application server written in Java, which runs on multiple platforms; *(ii)* a search server in which all sensor data is stored -in our design we use **Elasticsearch**[7], which provides a distributed search engine with an HTTP web interface and schema-free JSON documents; *(iii)* an orchestrator that ensures and manages communication paths between all containers. These three main components are virtualized by means of Docker containers; ensuring therefore an isolated environment together with many other benefits that will be clearer from the analysis of the different interactions that the gateway can perform with the rest of the network.

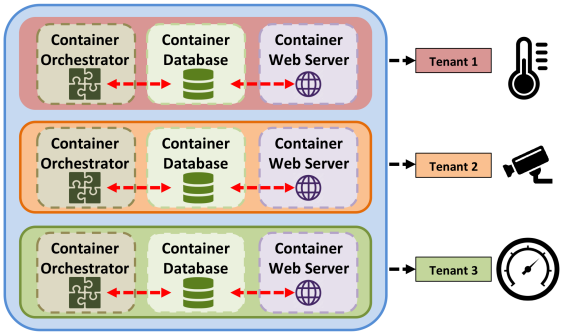


Fig. 2: The gateway can be shared between different tenants.

Ensuring *isolation* at the gateway allows also a *multi-tenant* platform that can be shared between different users (Fig.2). This characteristic turns out to be particularly useful in contexts in which the gateway has to keep the peculiarity of being a *vendor-independent* platform.

Finally, the flexibility given by the use of containers enlarges the potentialities of our platform in terms of backup functionality and capacity of storing the *network status* at given times.

[5]https://www.docker.io/
[6]http://wildfly.org
[7]https://www.elastic.co

Another main feature introduced in our gateway design is the *On-demand activation* of Docker containers, by means of a socket-activation framework. Figure 3 shows a practical use of this component. A remote user forwards the socket activation connection, which then goes through the proxy for activating the web server container. After the activation, the container can receive traffic from the sensors through the proxy. The example refers, in particular, to the activation of the web server container but it can be extended to the rest of the other containers.

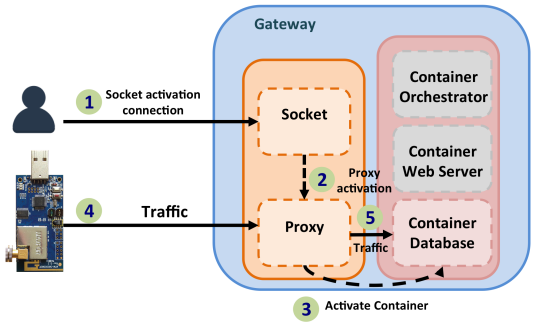


Fig. 3: Container activation through socket/proxy connection.

The implementation of the socket-activation framework allows a dynamic allocation of services in the gateway that can bring several benefits, for example in terms of energy efficiency. Indeed, the container instances can be dynamically allocated only when required, without being constantly active.

A subset of possible interactions that our gateway can keep with other network entities such as *Remote Users* and *Sensors* are detailed below.

For example, we want to analyze the simple case of a sensor that transmits its updated parameters to the gateway (Fig. 4a). We assume that the database container was previously activated by a remote user, and that the database containing the latest update data is stored within a cover Docker image. When the gateway receives updated information from the sensor, a Docker container is launched -via the socket/proxy activation- and the new values are stored into the database. Once this transaction is completed, the updated Docker image is saved locally in the gateway. The second interaction (Fig. 4b) can be considered consequently to the first. In this case a remote user aims to access data stored in the database. In this scenario, the user can easily download the latest updated Docker image -which is stored on the gateway- and then have access to data -this case is marked with 1 in the Figure 4b-. However, data can be read directly on the gateway from the Database Docker image. The last interaction (Fig. 4c) describes the case in which the gateway acts as a interface between remote users and sensors. In this case, we can consider the scenario of a user that wants to know the temperature measured in the "Room 105". In this particular case, thanks to the semantic annotation, the gateway knows the best node that can provide the information, then it forwards the request to the specific node.

IV. Performance Evaluation

The validation of our proposal covers two different aspects. First, we evaluate how a Raspberry Pi reacts -in terms of performance- to specific workloads generated by applications running within Docker containers. This is done with the aim to understand the impact of introducing virtualization on the aforementioned hardware. Then, we present a first performance evaluation of our gateway platform while performing specific tasks.

A. General Performance

To challenge specific hardware segments, in both the RPi2 and RPi3, we use software tools capable to generate different types of workloads. In order to estimate the overhead produced by the presence of a single or multiple containers running, we test *CPU*, *Memory*, *Disk I/O*, and *Network I/O* performance. The *native performance* -i.e. running the benchmark tool without including any virtualization layer- is used as a reference for comparison. Considering that in our gateway multiple functions are virtualized, by means of containers, our objective is to test the performance when concurrent (virtualized) instances are running.

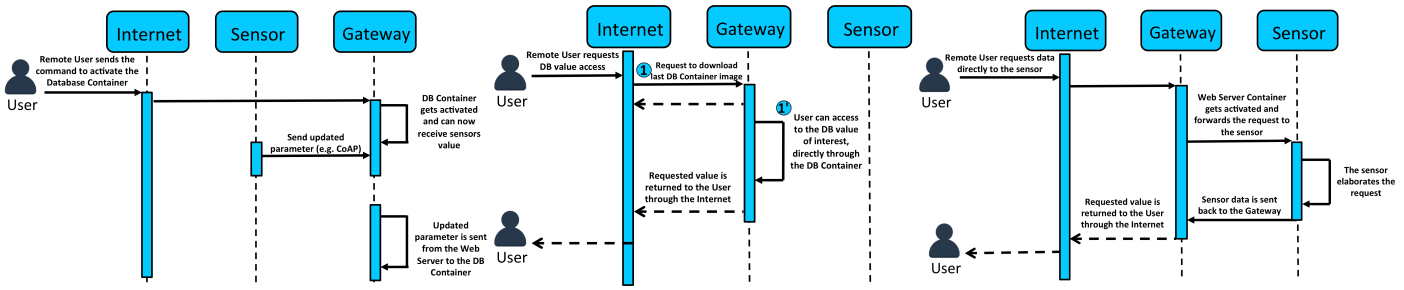
We test **CPU** performance with sysbench[8]. It executes a stress test designed to challenge the CPU by calculating prime numbers. The performance metric is the *execution time* (measured in seconds) -a lower execution time implies better performance-. Figure 5a shows the results coming from this test when up to four concurrent instances -native and virtualized- are running. We can mainly observe that the container engine introduces a negligible impact on the CPU performance in each single case. From the power consumption perspective, we can notice that the consumption of RPi3 is slightly higher compared to RPi2.

We use the Unix command *mbw*[9] to test the **Memory I/O** performance. The benchmark tool determines the available memory bandwidth by copying large arrays of data in memory, and performing three different tests (*memcpy*, *dumb*, and *mcblock*). Similarly to the previous case, native and container performance can be considered comparable (Figure 5b). RPi3 performance are approximately a third higher compared to the RPi2, at the expense of a minimal increase in the power consumption.

The benchmark tool fio[10] has been used in order to evaluate the **Disk I/O** performance. In particular, the test consists in performing sequential read/write operations against a 6GB file stored on the MicroSD card. Figure 5c shows sequential read and write performance averaged over 60 seconds using a typical 1 MB I/O block size. Docker introduces negligible overhead during the sequential read test. Both Raspberry Pi boards introduce a relevant overhead during the sequential write test almost 50% for RPi2 and 37% for RPi3.

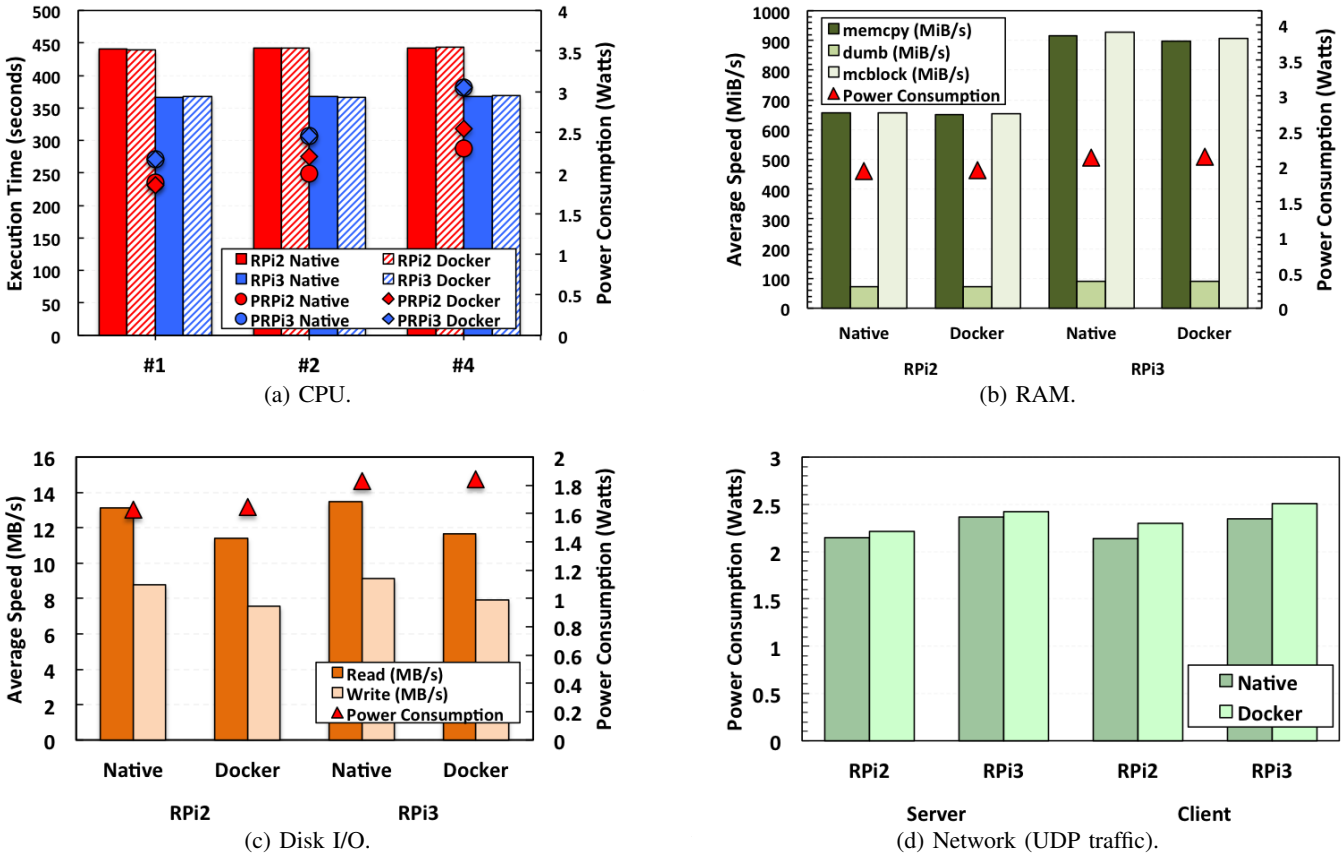
For the **Network I/O** analysis (Fig. 5d), we want to quantify the power consumption of the two Raspberry Pi boards, when the same amount of traffic -90 Mb/s of UDP traffic in our example- is sent/received. More in detail, this test aims to identify any power consumption increase due to the virtualization layer, both when the gateway acts as a server (receiving network traffic), and as a client (sending

[8]http://manpages.ubuntu.com/manpages/wily/en/man1/sysbench.1.html
[9]http://manpages.ubuntu.com/manpages/wily/en/man1/mbw.1.html
[10]http://linux.die.net/man/1/fio



(a) Gateway - Sensor. (b) User - Gateway. (c) User - Sensors.

Fig. 4: Gateway interactions with remote users and sensors.



(a) CPU. (b) RAM.

(c) Disk I/O. (d) Network (UDP traffic).

Fig. 5: General Performance Evaluation.

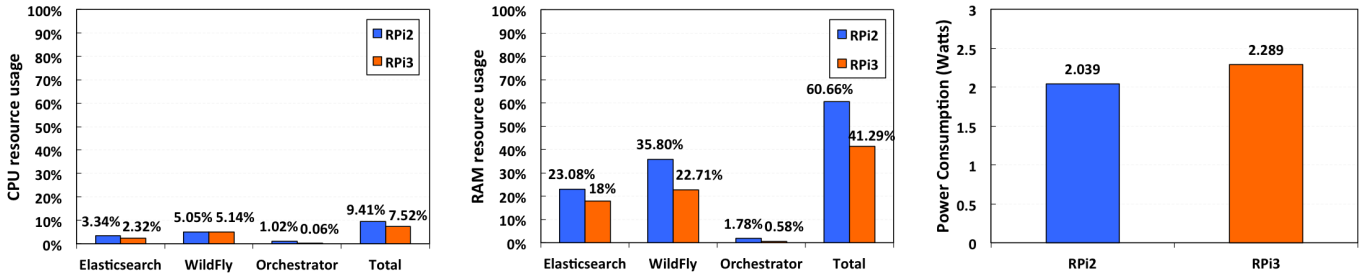
network traffic). From the results, we can observe that the power consumption of the Docker case slightly differs from the native execution. This implies that the power overhead generated by the *Virtual Network Interface Card* (virtual NIC) -when handling UDP traffic- is extremely low.

B. Gateway Performance

After demonstrating that running multiple Docker containers on top of a Single Board Computer does not impact the performance, in this second part of performance evaluation, we want to test our gateway in order to assess which board is more performing -by considering also the power consumption trade-off- while performing the same tasks. We assume that the *Database*, the *Orchestrator*, and the *Web Server* containers have been created -but not initialized- by a remote user.

The experiment conducted can be described as follows: in a first stage, the containers are booted up through the socket/proxy activation; a later phase characterized by the interaction sensors-gateway, in which a sensor sends data packets -13 bytes- every 60 seconds -according to ETSI specifications [11]-. As sensor platform, we used Maxfor nodes. These nodes embed a MSP430 as CPU and Texas Instruments CC2420[11] as radio frequency module; they are also equipped with temperature and light sensors.

[11] http://www.ti.com/product/CC2420



(a) CPU resource usage.

(b) RAM resource usage.

(c) Gateway power consumption.

Fig. 6: Gateway performance.

The nodes use Contiki-OS[12] as operating system and in particular we flashed their firmware with CACHACA (Confident-based Adaptable Connected objects discovery to HArmonize smart City Applications) a ranking mechanism that we proposed in [12]. By running this algorithm, sensor nodes can evaluate and classify neighbors and the services that they offer (e.g., temperature, light, humidity, etc). CACHACA is based on a rule-based fuzzy inference system and the use of physical parameters -such as the Received Signal Strength Indication (RSSI)- in order to rank neighborhood and services.

The overall duration of the test is one hour and two minutes, where the first two minutes are used to create and initialize the containers. In this span of time, different performance metrics have been stored; the Raspberry Pi power consumption, and the resources -CPU, Memory- employed by each running container, thanks to the docker stats command.

Figure 6 shows the outcome of this evaluation. In Figure 6a the CPU resource usage of each single container is shown. It can be observed that the total amount of employed CPU resources does not exceed 10%. The web server is the component that produces the higher CPU utilization (approximately 5%). Raspberry Pi3 is the board that shows optimized performance compared to Raspberry Pi2. From the Memory performance perspective (Fig. 6b), we can mainly notice a relevant resource usage optimization in the RPi3 for each component container, which is then reflected in the overall percentage. Again, the web server is the container more greedy in terms of resources. In both cases analyzed previously, the *Orchestrator* is the container that requires less resources. Another clear aspect coming from this evaluation is the better efficiency of RPi3 compared to RPi2. However, in the comparison RPi2/RPi3 the trade-off between power consumption and performance has to be considered. In Figure 6c, the average power consumption of the two devices under test is depicted. The better RPi3 performance are achieved at the expense of a slight higher power consumption -approximately 12% higher-.

V. Conclusion

In this paper, we introduced a lightweight Edge Gateway-as-a-Service and its architecture. Through the use of container vitrualization technologies we satisfied many platform requirements. Our proposal has been validated by means of a wide performance evaluation. The main insights coming from the validation are two-fold: *(i)* we have demonstrated that employing virtualization technologies on top of constrained devices has an almost negligible impact in terms of performance; *(ii)* the low resource usage of our gateway confirms the lightweight characteristics of our design, revealing promising results in terms of scalability and energy efficiency. The GaaS described in this paper is targeted for being employed in different IoT contexts -such as, smart home, buildings, farms, etc.- thanks also to the use of semantic technologies that enable interoberability among heterogeneous devices.

[12]http://www.contiki-os.org

References

[1] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything," *CISCO white paper*, pp. 1–11, 2011.

[2] R. Petrolo, V. Loscrì, and N. Mitton, "Towards a Smart City based on Cloud of Things, a survey on the smart city vision and paradigms," *Transactions on Emerging Telecommunications Technologies*, 2015.

[3] R. Petrolo, R. Morabito, V. Loscrì, and N. Mitton, "The design of the gateway for the cloud of things," *Annals of Telecommunications*, pp. 1–10, 2016.

[4] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," in *Proceeding of the International IEEE Conf. on Mobile Services*, New York, New York, United States, Jun. 2015.

[5] Z. Wu, T. Itl, T. Tang, C. Zhang, Y. Ji, M. Hmlinen, and Y. Liu, "Gateway as a service: A cloud computing framework for web of things," in *Proceeding of ICT - 19th International Conf. on Telecommunications*, Jounieh, Lebanon, Apr. 2012.

[6] B. I. Ismail, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *Proc. of ICOS - International IEEE Conf. on Open Systems*, Malacca, Malaysia, Aug. 2015.

[7] R. Morabito and N. Beijar, "Enabling data processing at the network edge through lightweight virtualization technologies," in *Proc. of SECON - International IEEE Conf. on Sensing, Communication, and Networking - Workshops*, London, United Kingdom, Jun. 2016.

[8] O. Novo, N. Beijar, M. Ocak, J. Kjllman, M. Komu, and T. Kauppinen, "Capillary networks - bridging the cellular and iot worlds," in *Proc. of WF-IoT - 2nd IEEE World Forum on Internet of Things*, Milan, Italy, Dec. 2015.

[9] B. Smith, "ARM and Intel battle over the mobile chip's future," *Computer*, vol. 41, no. 5, pp. 15–18, 2008.

[10] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *Proc. of IC2E - International IEEE Conf. on Cloud Engineering*, Tempe, Arizona, United States, Mar. 2015.

[11] ETSI, "Technical report 103 055," Technical report, 2011. [Online]. Available: http://www.etsi.org/deliver/etsi_tr/103000_103099/103055/01.01.01_60/tr_103055v010101p.pdf

[12] R. Petrolo, V. Loscri, and N. Mitton, "Confident-based Adaptable Connected objects discovery to HArmonize smart City Applications," in *Proc. of IFIP WD - Wireless Days*, Toulouse, France, Mar. 2016.